EXPERIMENT TWO

1) Water Jug Problem

Problem Statement

Objective:

1. Given two jugs — one with a capacity of 3 liters and another with a capacity of 4 liters — and an unlimited water supply, the goal is to obtain exactly 1 liter in the 3-liter jug.

Legal Operations:

1. Completely fill a jug.
2. Completely empty a jug.
3. Transfer water from one jug to another (until one is empty or the other is full).

Representation:

1. Use an ordered pair (x, y) where:
2. x = amount of water in the 3-liter jug
3. y = amount of water in the 4-liter jug Initial state is (0, 0), and the goal is to reach any state (1, y).

```
from collections import deque

# Function to check if the state has already been visited
def is_visited(state, visited):
    return state in visited

# Function to perform BFS and find path to (1, y)
def water_jug_problem():
    max_x, max_y = 3, 4  # Jug capacities
    start = (0, 0)
    goal_x = 1
    visited = set()
    queue = deque()
    queue.append((start, [start]))  # (current_state, path_to_state)

    while queue:
        (x, y), path = queue.popleft()

        if x == goal_x:
            print("Path to reach (1, y):")
            for state in path:
                print(state)
            return

        visited.add((x, y))

        # All possible operations
        possible_moves = [
            (max_x, y),  # Fill 3-liter jug
            (x, max_y),  # Fill 4-liter jug
            (0, y),      # Empty 3-liter jug
            (x, 0),      # Empty 4-liter jug
            # Pour 3 -> 4
            (x - min(x, max_y - y), y + min(x, max_y - y)),
            # Pour 4 -> 3
            (x + min(y, max_x - x), y - min(y, max_x - x))
        ]

        for new_state in possible_moves:
            if not is_visited(new_state, visited):
                queue.append((new_state, path + [new_state]))

    print("No solution found.")

# Run the function
water_jug_problem()
```

```
Path to reach (1, y):
(0, 0)
(0, 4)
(3, 1)
```

```
(0, 1)
(1, 0)
```

2)Missionaries and Cannibals Problem

Problem Statement

Objective:

1. Three missionaries and three cannibals need to cross a river using a boat that can carry a maximum of two people. The boat cannot cross the river by itself.

Constraints:

1. At no point should cannibals outnumber missionaries on either side of the river.
2. The boat can carry 1 or 2 people at a time.
3. Either missionaries or cannibals or both can operate the boat.

Representation:

1. Each state is represented as a 6-tuple:
    - (ML, CL, MR, CR, boat_side, path) where:
    - ML, CL: Missionaries and Cannibals on the left bank
    - MR, CR: Missionaries and Cannibals on the right bank
    - boat_side: "left" or "right"
    - path: Sequence of moves to reach the state (optional, for printing solution)

```
from collections import deque

def is_valid_state(ml, cl, mr, cr):
    # Check for validity of state
    if ml < 0 or cl < 0 or mr < 0 or cr < 0:
        return False
    if ml > 0 and ml < cl:
        return False
    if mr > 0 and mr < cr:
        return False
    return True

def get_successors(state):
    ml, cl, mr, cr, boat = state
    successors = []
    if boat == "left":
        # Try all combinations of 1 or 2 people crossing from left to right
        moves = [(1,0), (2,0), (0,1), (0,2), (1,1)]
        for m, c in moves:
            new_ml, new_cl = ml - m, cl - c
            new_mr, new_cr = mr + m, cr + c
            if is_valid_state(new_ml, new_cl, new_mr, new_cr):
                successors.append((new_ml, new_cl, new_mr, new_cr, "right"))
    else:
        # Try all combinations of 1 or 2 people crossing from right to left
        moves = [(1,0), (2,0), (0,1), (0,2), (1,1)]
        for m, c in moves:
            new_ml, new_cl = ml + m, cl + c
            new_mr, new_cr = mr - m, cr - c
            if is_valid_state(new_ml, new_cl, new_mr, new_cr):
                successors.append((new_ml, new_cl, new_mr, new_cr, "left"))
    return successors

def solve_missionaries_and_cannibals():
    start = (3, 3, 0, 0, "left")
    goal = (0, 0, 3, 3, "right")
    visited = set()
    queue = deque()
    queue.append((start, [start]))

    while queue:
        current_state, path = queue.popleft()
        if current_state[:5] == goal:
            print("Solution found:")
            for step in path:
                print(step)
            return

        visited.add(current_state)
        for succ in get_successors(current_state):
```

```
            if succ not in visited:
                queue.append((succ, path + [succ]))

    print("No solution found.")

# Run the function
solve_missionaries_and_cannibals()
```

```
Solution found:
(3, 3, 0, 0, 'left')
(3, 1, 0, 2, 'right')
(3, 2, 0, 1, 'left')
(3, 0, 0, 3, 'right')
(3, 1, 0, 2, 'left')
(1, 1, 2, 2, 'right')
(2, 2, 1, 1, 'left')
(0, 2, 3, 1, 'right')
(0, 3, 3, 0, 'left')
(0, 1, 3, 2, 'right')
(1, 1, 2, 2, 'left')
(0, 0, 3, 3, 'right')
```